Extending an IMS Client with Peer-to-Peer Content Delivery

J. Fiedler[1], T. Magedanz[2], and J. Müller[1]

[1] Fraunhofer FOKUS, Kaiserin-Augusta-Allee 31, 10589 Berlin
{Jens.fiedler,Julius.Mueller}@fokus.fraunhofer.de
[2] Technische Universität Berlin, FR 5-14, Franklinstr 28/29 10587 Berlin
tm@cs.tu-berlin.de

Abstract. The increasing demand for mobile applications implies an increase in service availability and content delivery capacities across the networks. Peer-to-peer technologies have proven to be able to deliver media in an effective way to the end user. In this paper, we analyze and describe the necessary extensions and functionalities, which are needed to enable Peer-to-peer content delivery in an IMS client, namely the MONSTER framework. A special focus is directed to the interoperation between existing functional elements and newly developed peer-to-peer components.

1 Introduction

In today's telecommunication world, several aspects of the service are important to the customer. These are e.g. mobility, service availability and diversity of services. The IP multimedia subsystem (IMS) [11] has proven to be an architectural framework, which grants exactly those aspects to the customer. IMS enables the standardized access to IP based services out of different types of networks.

Due to its All-IP approach, the IMS is a key enabler for converged internet services, not limited to Voice-over-IP (VoIP) communication. Today, we see an increasing demand for additional services on the mobile phones, like television, gaming, community services, etc.

Today, two basic ways exist to gain access to multimedia content. One way is by using a media server (central streaming), the other by joining a Peer-to-peer (P2P) content delivery overlay (distributed streaming).

In this paper, we present a generic blueprint for a user client, suitable for IMS communication and P2P media delivery. As initial point we have chosen an IMS client, because IMS clients are supposed to have similar structures, while P2P clients heavily depend on the intended purpose, i.e. file-sharing, streaming, computing, etc. Therefore, the presented approach can be adapted easily to different existing IMS clients.

The layout of this paper is as follows. In chapter 2, we shortly discuss IMS functionalities from the client point of view and Peer-to-peer media delivery technologies. Chapter 3 will give a short motivation, why it is suitable to implement P2P functionalities in an IMS client. In chapter 4, we introduce necessary P2P

© Institute for Computer Science, Social Informatics and Telecommunications Engineering 2009

functional blocks and present a generic way how to extend an IMS client with them. In chapter 5, we give an example by explaining, how a concrete IMS client is going to be extended for P2P media delivery. Finally, in chapter 6 we draw some conclusions and give an outlook to the future work.

2 Background

In this section we will give a brief introduction in IMS and P2P services.

The **IMS** is an architectural framework for delivering IP-multimedia to mobile users. It was originally designed by the wireless standards body 3rd Generation Partnership Project (3GPP), and was intended to lead a way for mobile networks beyond GSM. Its original formulation (3GPP R5) represented an approach to delivering "Internet services" over GPRS. This vision was later updated by 3GPP, TISPAN by requiring support of networks other than GPRS, such as Wireless LAN and fixed line. The IMS network system consists of different functions, interacting over standardized interfaces (reference points), which form one IMS administrative network.

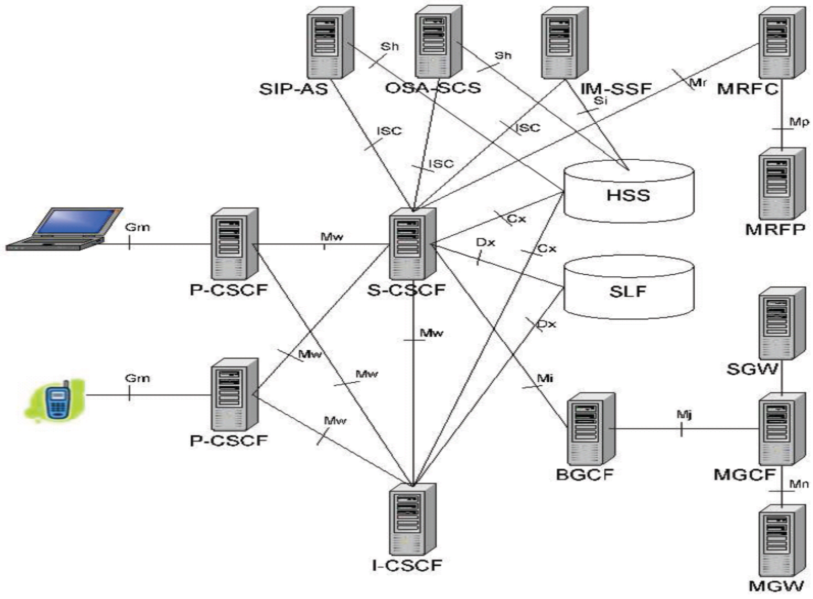

Fig. 1. IMS Components and reference points

An IMS-function is not necessarily identical to a node (hardware box). An implementer is free to combine 2 or more functions in one single node, or to spread a single function over multiple nodes. Each function can also be present multiple times in a single network, for load balancing, availability purposes or organizational issues. Reference points are realized by standardized protocols, like the session initiation protocol (SIP) [2] or DIAMETER [3]. Fig. 1 illustrates the most relevant IMS functions, the position of the user client and the related reference points between them.

For P2P services, we examine P2P distributed hash tables (DHT), P2P media-streaming and P2P file sharing. All P2P services are aiming on a distribution of data, may it be key-value pairs, stream chunks or file pieces. We do not focus on indexing here, i.e. the way how nodes discover content. We assume that this has been properly done before the envisaged content distribution, i.e. that nodes know which overlay to join for a specific content.

Nevertheless, there are two operations, which all P2P algorithms have in common. These are *join* and *leave*. The *join* operation is performed by a node that wishes to become part of the corresponding overlay. It results in a message, which is sent to a particular node in the overlay, the bootstrap node for the joining node. The way this bootstrap node is detected depends highly on the associated P2P network. Lists of well-known nodes, broadcasting, etc. are suitable techniques to detect a bootstrap node. After a successful *join* operation, the performing node is part of the overlay and knows a relevant subset of nodes, its neighborhood. The *leave* operation is to be used when a node orderly leaves the overlay. As P2P networks are considered to be self-healing, this operation must be understood as "the polite way" to leave an overlay. The remaining P2P network will continue to function even if a leaving node does not issue a *leave* message to the overlay.

For a **P2P DHT**, the additional operation *putkey* and *getkey* are used. The underlying DHT algorithm is not of relevance here, it can be e.g. CHORD [5] or something similar. The *putkey* operation accepts a key and a value as its arguments and decides where in the overlay this information is going to be stored. It then issues a message either directly to the storing node, or to a node in its neighborhood, which then either routes the message further to the storing node or returns the address of it, optionally by recursively querying other nodes first. After the storing node received the *putkey* message, it will store the key with its value. The opposite operation to *putkey* is the *getkey* operation, which retrieves a value to a given key, basically in the same way, as the *putkey* operation stores it.

For **P2P streaming**, many approaches exist, aiming on multiple different aspects of live streaming. A comprehensive comparison can be found in [1]. The operations, we are focusing on are *sendstream*, *receivestream* and *requeststream.* We assume that P2P streaming is realized by a distribution graph, which is thinner to the source, and wider, the further away in terms of overlay nodes from the source a node resides. The graph is intended to be, but is not necessarily a tree, as nodes (children) can receive partial streams from different senders (parents). The *requeststream* operation will select a stream from one or more senders. It will also define, how the stream is to be sent in terms of which parts of the stream (interleave), quality, etc., if this is not implicitly done by joining the specific overlay. The *sendstream* operation will send the selected stream in the requested way to the sink. The *receivestream* operation will receive the stream from one or multiple senders and optionally re-assemble the partial streams to a playable media stream. This is necessary if the stream consists of multiple chopped sub-streams, which were received from different sources. The *requeststream* operation needs to be performed only once per stream.

For **P2P filesharing** there are three operations which are required. They are *requestblock*, *sendblock* and *receiveblock*. The *requestblock* operation is used to instruct another node to send the specified part or piece of a content (file) to the requestor. The selection algorithm is independent from this operation. The *requestblock* operation must be performed either for each block, or for a group of blocks, depending on the distribution policy of the P2P algorithm. The *sendblock* operation is used to send the requested block to the requestor. Here it is dependant on the P2P technique, whether the block is transferred directly to the requestor or indirectly by routing it over a set of other nodes, e.g. super-nodes, or anonymizing nodes. The *receiveblock* operation receives a piece of content and places it at the correct location in the local storage for that content.

As an IMS client uses the SIP for signaling with the IMS core and its components, it must have a SIP stack inside. This makes it feasible to use a SIP oriented protocol at least for the non-media part of P2P communication. Here, the P2PSIP comes in handy, which is currently developed by an IETF group [6]. P2PSIP focuses mainly on managing P2P overlays using the SIP. It extends SIP by defining new header-fields and is therefore fully compliant to the original SIP and can be expected to be supported by traditional SIP stacks.

3 Motivation

A single IMS client is exactly that, a client for the services, which are offered by the different IMS operators. By adding P2P functions, new aspects for the customer as well as for the providers and operators arise. Ongoing research has also discovered the benefits of combining P2P and IMS on different layers [9] and also already for static media [10].

It has already been discussed in [4], that an external DHT, which could be a client based P2P network, could be used for storing contact addresses for the failure case of the central architecture. Clients could then store and retrieve their contacts in a client based distributed hash table (DHT) as long as the central architecture is not available. Naturally, this results in a security challenge against falsification of such contact records. This is currently discussed in the P2PSIP group of the IETF [7].

When talking about P2P media delivery, we must distinguish between 3 basic types of multimedia content and content consumption. These are Live-streaming, video on demand and static content. They differ in their real-time criticalness, which is very high for live-streaming as it does not allow a big pre-buffering, followed by video on demand, which does allow pre-buffering. Static content is basically classic file sharing, which has no hard real-time requirements, it is finished when it is finished, or when the user decides to cancel the download, because it takes too long for him. Hence, P2P content delivery can help to unburden media servers or make them superfluous.

This is also the reason, why user-generated content experiences better support by P2P content delivery. A content generating user node can be seen as media server with extremely small banded upload capacities. Making it stream to every content sink is impossible by concept. The “IMS-way” would be to stream the content over a media server, resulting in the known load and availability problems of media servers.

The P2P approach here makes it much easier for a user to place its own content in the network.

4 Relevant Extensions

The question of how an IMS client extension should look alike in a generic way, refers to the underlying question of how the architecture of a suitable IMS client should look like. The requirements for the architecture of an IMS client contain a modular and extensible design and should be conforming to the relevant standards to ensure interoperability with other IMS components. The architecture should roughly be able to be subdivided into at least three abstract layers. Additional requirements to the IMS client are, for instance: on top a presentation layer, in the middle an application layer and at the bottom a service layer.

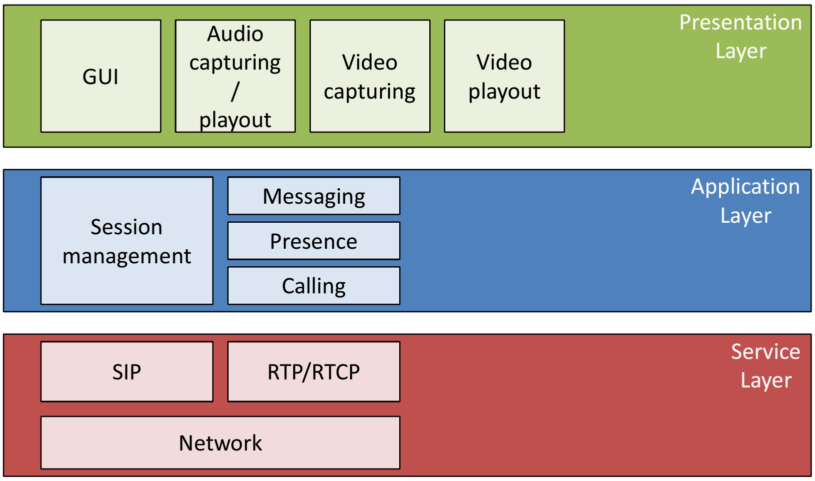

Fig. 2. Generic IMS Client Layer Model

The presentation layer is responsible for the interaction with the user. It should be designed to be easily extensible, in order to implement new features and display options. The presentation layer has to handle different endpoints like mobiles or desktops concerning the presentation technology and has to provide a suitable adaption of it.

The application layer enables the client functionalities like calling or messaging. Events coming from the user interface have to be interpreted and translated into program operations to provide the user interaction with the client. These operations then will trigger services in turn.

The service layer provides core services of the IMS client and provides an API to the upper layer for e.g. SIP-messaging for IMS-core interaction.

The described model is depicted in Fig. 2.

The following example maps the messaging service to the three presented layers. The user A sends a message to user B with an IMS client. Thereby the presentation layer interacts with the user over e.g. a text field for message content, the receiver selection and the send button. The presentation layer also includes the media capture and play-out capabilities. The application layer provides methods to write and store a message, send a message via the IMS core to the receiver(s) and handles incoming messages. The service layer provides the API to use the underlying SIP stack to perform the necessary SIP operations to send the message.

If we now want to extend the client with P2P functionalities, a protocol for the communication between the peers is needed. All P2P tasks e.g. management of the overlay or DHT functionalities like *put* and *get* are message based. Since all participating peers are instantiated through IMS clients which uses the SIP protocol, it is assumed that the IMS client architecture comprised a SIP stack.

One important technique of the domain software engineering is to reuse existing parts. In our case the existing SIP-stack should be reused and extended to provide an API for using P2PSIP (s.a.) operations.

The obvious challenges of extending a SIP-stack with the methods of P2PSIP base upon the fact that SIP uses the unique SIP-URI to address a recipient whereas P2PSIP in contrast uses the IP and port information of the client. The P2PSIP functionality should be realized in the service layer to provide its service to the higher layer.

There are task specific functionalities which could be grouped as the following functional blocks: general functions, overlay management, DHT, media streaming and file sharing.

The general P2P functions are:

- *Determine the next hop*, which realization depends on the used P2P algorithm. The presence of NAT lead to the fact that there occur cases in which the sending peer isn't able to reach the requesting peer directly and has to route packages indirectly to the requesting peer. All following tasks assume the appliance of this function to ensure a more reliable communication.
- A method for *generating hashes* uses a known cryptographic hash function like (MD5) to generate hashes e.g. out of the unique IP address and port combination of a peer or the name of a given value.

The overlay management functions are:

- A peer uses the *join* method to participate in an overlay. This task has to be divided in the active and passive case. The active part would be the bootstrapping, where a new peer (*N*) contacts a known bootstrap peer (*B*) to receive an overlay position with the contact information of its successor peer (*S*) and its predecessor peer (*P*).
- The passive part of joining an overlay would be
- The task of the bootstrapping peer (*B*) describes the passive part of joining an overlay, in which peer (*B*) has to organize the arrival of peer (*N*). The resulting tasks are organizationally to maintain the overlay structure after the new peer (*N*) has finished the process of joining the overlay. The two adjacent peers (*S*) and (*P*) of (*N*) have to be informed of its arrival, what results in the modification of their routing tables in case of a DHT based algorithm like Chord.

- To *leave* a network in a polite way means that the leaving peer (*L*) informs all relevant peers and to pass optionally stored values to peers which are responsible, when (*L*) leaves the overlay.

The DHT functions are:

- The method *lookup* maps a hash to an IP address and port tupel and is able to associate a key with a peer.
- Since everything has its place in a DHT, each value has its defined place which depends on its referring hash. To store a value in the way that other peers are able to find it, a hash function is used to map the name of the value into an unique hash value: the key referring this value. The number space of the keys and that of the peerIDs were equal, which facilitates to assign a value with the help of its key to a peer.
- To store a value in the DHT, a hash of the value has to be generated which refers to peer (*R*). Peer (*R*) has to be determined with the use of the hash function in combination with the *lookup* method. Finally a reference to the value has to be send from the initiating peer to the peer (*R*).
- The *get* method is used to retrieve a value out of the overlay from other peers. Thereby a hash of the requested value identifies the referring peer (*R*). The requesting peer performs a lookup of the value and retrieves the contact information of the peer (*S*), which stores the value. Now the requesting peer is able to address peer (*S*).

The streaming functions are:

- The method *send* is used to send a particular part of a stream to a requesting peer.
- The method *request_stream* requests a specific part of a stream from a parent node.
- The method *receive* stores all requested parts of the stream to re-assemble them to a single stream. The parts needed to be encoded independent of other parts, that they are able to be consumed in pieces.
- The method *receive_request_stream* handles an incoming request for a part of a stream. Either the request is allowed or it will be rejected. In case of free resources of the requested peer the demanding peer will be delivered with the requested part of the stream. In case of high utilization or missing capabilities a cancelation will be send to the requesting peer.

The file sharing methods are:

- The method *request_block* is used to request a specific part of a whole file from a known source.
- With the help of *receive_block* a requested part can be received.
- The interaction of the peers needs the *receive_request_block* method which handles block requests which are either served to the demanding peer in case of free recourses or rejected in case of problems.
- The method *send_block* finally sends a block to the requesting peer.

To realize the mentioned P2P functions in the presented IMS client architecture, the following extensions need to be performed to the existing layers. Fig. 3 illustrates the distribution of the functional blocks over the different layers.

The *presentation layer* should be extended with two control mechanisms. First the control of P2P streams like: select, start, stop, pause of a requested object have to be handled. The second part covers the content presentation with VRC controls like: start, stop and pause. As these changes affect only the GUI, they are not depicted in Fig. 3.

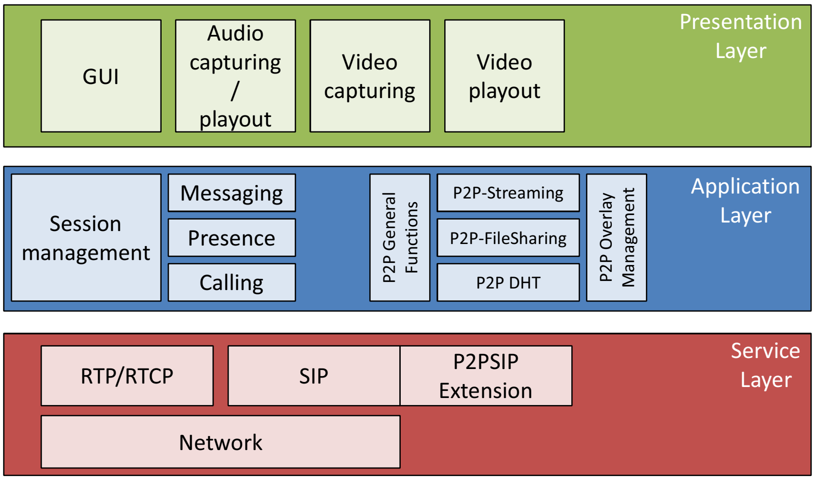

Fig. 3. Extended Generic IMS Client Model

The tasks of the *application layer* are extended with the previously introduced functional blocks: general functions, overlay management, DHT, media streaming and file sharing.

The *service layer* is extended with P2PSIP as a new service. This is attached to the existing SIP stack and needs no extra network operation by itself, but provides an API to the application layer.

5 Case Study: MONSTER

The Fraunhofer FOKUS MONSTER IMS client aims the rapid development and prototyping of NGN and internet applications.

The most motivating factors to use this client are its modular architecture, its platform independence with a pure java implementation plus it is deployable on target platforms like mobiles, laptops and desktops as well. The MONSTER client is standard conform and extended the JSR281 specification which provides a high-level API to access IP Multimedia Subsystem (IMS) services. This API hides IMS technology details and exposes service-level support to enable easy development of IMS applications. In February 2009 MONSTER will be offered as free-to-use. It will provide several free basic functions and will be extensible through an open API which provides interfaces to enhance and extend its functionalities [8].

Since the architecture of MONSTER is modular, it is possible to divide its architecture into the three main parts of a presentation, an application and a service layer. Thereby it is possible to apply the presented P2P extension on it. The layers have to be modified as follows.

The presentation layer of MONSTER provides interaction with the P2P module to perform the requested tasks of the user. These consist of the control of the requested content and the playback handling with the common video recording functions.

The functional blocks from chapter 4 (general, management, DHT, streaming, file sharing) have to be aggregated to a P2P module which is needed to be adapted to the application layer.

The domain of event handling needed to be extended, too. Notifications are redirected to the P2P module for sending or receiving P2PSIP messages or their retransmission.

The functional blocks are message based and needed to reply incoming requests automatically. Therefore an instance is needed, which is able to answer requests automatically, by performing the required P2P specific tasks.

The used SIP-stack of MONSTER has to be modified and extended to provide P2PSIP in the service layer. To reuse the SIP stack, a type P2PSIP message has to be derived from the type SIP message.

Since each client communicate only directly to the PCSCF on a fix port over SIP, P2PSIP has to address clients dynamically on different ports and IP addresses.

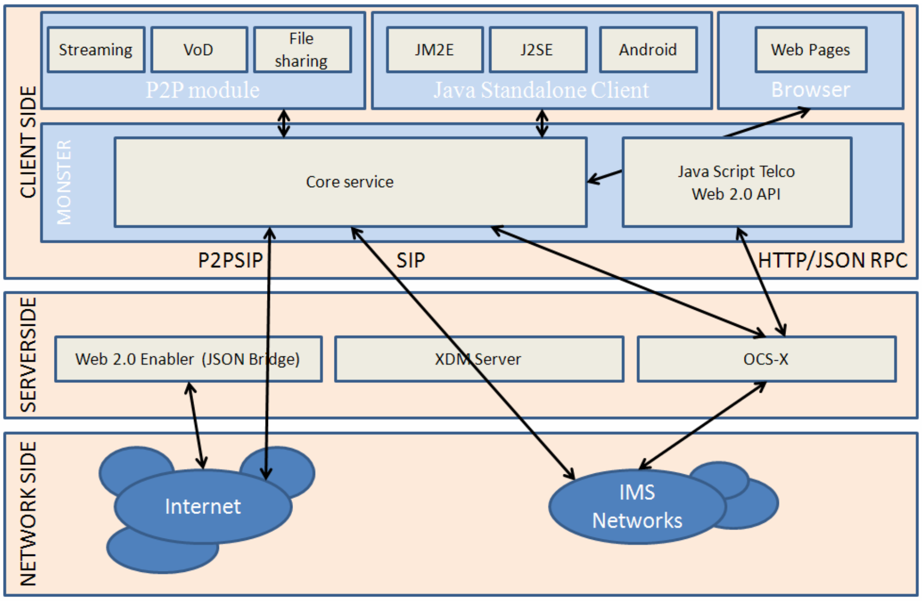

Fig. 4. MONSTER architecture

Fig. 4 depicts the interaction of the P2P module with the MONTER client. The P2P module is connected with the core service of monster as well as directly with the internet. Additional features of the MONSTER client are presented like the OCS-X Parley Interface, which is used for IPTV and VoD. The Web 2.0 Enabler allows the aggregation of web feeds like news, weather, Flickr, Facebook or Google APIs. The XDM Server enables service configuration like group management or presence.

6 Conclusions and Future Work

In this Paper, we have presented a generic approach to extend IMS clients with P2P functionalities. As an example, the necessary extensions to the Fraunhofer FOKUS MONSTER IMS client framework have been explained and depicted. Nevertheless, while exploring the potential of P2P and IMS coupling, it became obvious that many features require additional support of IMS components. As an example, authentication needs to be named along with the possibility to use the underlying NGN components for quality-of-service (QoS) control between peers. Also, the creation of topology aware overlays can be supported with knowledge from IMS core components. A topology aware overlay can drastically reduce latency and cross-network traffic, resulting in a reduction of costs for avoiding redundant traffic.

Also content integrity and security need to be addressed. These are major problems of open P2P systems, where content poisoning and copyright infringements are a daily appearance. Also more complex approaches to detect copyrighted content could be considered, like distributed content fingerprint detection.

Another thing is the question "Do other P2P services require other operations?" It is very likely, that e.g. a P2P community management will require different operations than those which are presented here. Nevertheless, the presented approach can be easily extended to integrate new operations, due to its modular construction.

Acknowledgements

Many thanks go to the MONSTER developer team at the group for Next Generation Networks Infrastructures (NGNI) at the Fraunhofer FOKUS Institute in Berlin, Germany for inspiring discussions.

The work presented in this paper is related to the ongoing EU project VITAL++. VITAL++ is a Specific Targeted Research Project (STREP) supported by the European 7th Framework Programme, Contract number ICT-2-1.6-224287, Project starting date 1st June 2008 (duration 30 months).

References

[1] Magharei, N., Rejaie, R., Guo, Y.: Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In: 26th IEEE International Conference on Computer Communications (INFOCOM), pp. 1424–1432. IEEE Press, Anchorage (2007)

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A.R., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: session initiation protocol, RFC 3261, IETF (June 2002)

[3] Calhoun, P., Loughney, J., Guttman, E., Zorn, G., Arkko, J.: Diameter Base Protocol. RFC 3588, IETF (September 2003)
[4] Singh, K., Schulzrinne, H.: Using an External DHT as a SIP Location Service, Columbia University Technical Report CUCS-007-06, New York, NY (Feb. 2006)
[5] Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In: SIGCOMM (2001)
[6] Bryan, D., Matthews, P., Shim, E., Willis, D., Dawkins, S.: Concepts and Terminology for Peer to Peer SIP, draft-ietf-p2psip-concepts-02, IETF, July 7 (2008)
[7] Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., Schulzrinne, H.: Resource Location And Discovery (RELOAD), draft-ietf-p2psip-reload-00, IETF, July 11 (2008)
[8] Bachmann, A., Motanga, A., Magedanz, T.: Requirements for an extendible IMS client framework. In: ACM International Conference Proceeding Series, vol. 278 (Feburary 2008)
[9] Liotta, A., Ling, L.: The Operator's Response to P2P Service Demand. In: Communications Magazine. IEEE, Los Alamitos (2007)
[10] Fiedler, J., Magedanz, T., Menendez, A.: IMS secured content delivery over peer-to-peer networks. In: Proceedings of SIGMAP 2007, Spain, July 28-31, 2007, pp. 5–12. INSTICC Press, Portugal (2007)
[11] TS 23.228, IP multimedia subsystem (IMS), 3GPP (2006)